applications include secure video devices, tactical handheld radios, front-ends for Intelligent Surveillance and Reconnaissance (ISR), secure communications modules, sensors, or telemetry units.

Configuring a computer system to include fully independent instruction and data spaces can help improve the security of the computer system. Using byte-code transformation ("Harvardization") or encryption, systems can reduce the attack surfaces that exist as a result of shared instruction and data spaces or buses used in many computer systems. Encryption of the instruction, data, and microcode spaces can help thwart tampering and reverse engineering. The pre-execution of Harvardized code can help disable byte-code based attacks, such as code injection attacks.

Systems described herein can be configured to execute legacy Von Neumann architecture instruction sets in a system with a Harvard architecture (e.g., a pure Harvard architecture). A pure Harvard architecture means a system where instructions and data formatted for execution in a Von Neumann environment can be transformed to a form executable by a system with a Harvard architecture. The transformed instructions and data can be encrypted, such as by using a relatively low-latency encryption algorithm and low-latency decryption algorithm (e.g., combinatorial algorithms that can combine arithmetic and logical operations such as a one-time pad algorithm or other algorithm). The encryption and decryption algorithm used is flexible and can be chosen based on the needs of the application. The decrypted instructions and data can be transmitted to a Transformation Execution Engine (TXE) which can be configured to execute legacy instruction sets and provide anti-malware and anti-tamper security. Systems described herein can be implemented in, for example, an embedded, real-time computer architecture.

FIG. **1** shows an example of a Harvard architecture **100**. The Harvard architecture **100** can include a Central Processing Unit (CPU) **102**, physically separate or independent instruction memory **104** and data memory **106**, and Arithmetic Logic Unit (ALU) or Floating Point Unit (FPU) **108**, and Input/Output (I/O) **110** lines to peripheral devices. The CPU can include the ALU or FPU **108**, a Computer Control Unit (CCU) **107**, and other control logic.

The instruction memory **104** can be read only memory and the data memory **106** can be read-write memory. The instruction memory **104** can be an operating system (OS) or an application memory. The data memory **106** can be an application memory. The instruction memory **104** and the data memory **106** can be physically separate or independent, such as to include no common or shared signal paths. The instruction memory **104** and the data memory **106** can be communicatively or electrically coupled to the CPU **102**. The instruction memory **104** can include control logic. The data memory **106** can include read/write and control logic.

The ALU or FPU **108** can be configured to perform mathematical or other operations on data received from the CPU **102**. The CPU **102** can instruct the ALU or FPU **108** which operations it is to perform and which data the ALU or FPU **108** is to perform the operations on.

The I/O **110** can be electrically or communicatively coupled to the CPU **102** and a peripheral device, such as a sensor, video, transceiver, Geographical Positioning System (GPS), or other peripheral device.

FIG. **2** shows an example of a computer system **200** with a Harvard architecture that is configured to be tamper and malware resistant. The computer system **200** can include a harvardizer **212**, an encryptor **214**, the instruction memory **104**, the data memory **106**, a decryptor **216**, a combination encryptor and decryptor **218**, and the CPU **102**.

The harvardizer **212** can receive data and instructions configured for use in a computer system with a Von Neumann architecture or another architecture where the data and instructions are comingled or not independent. The harvardizer **212** can be configured to separate the comingled data and instructions. For example, if the harvardizer **212** receives a bit string "0101010111110000" that is intended to be used in an eight-bit Von Neumann architecture, then the harvardizer **212** can parse the string into the instruction "01010101" and the data "11110000". The harvardizer **212** can be configured to determine how the data received is configured so that it can accurately parse the data and the instructions. The harvardizer **212** can send the parsed (e.g., separated) data to the data memory **106** and the instructions to the instruction memory **104**.

The optional encryptor **214** can encrypt the parsed data or instructions from the harvardizer **212** and send them to the encrypted data to the data memory **106** and the encrypted instructions to the instruction memory **104**. The optional decryptor **216** can receive encrypted instructions from the instruction memory **104** and send decrypted instructions to the CPU **102**. The optional encryptor and decryptor **218** can receive encrypted data from the data memory **106** and send decrypted data to the CPU **102**. The optional encryptor and decryptor **218** can receive data from the CPU **102** and send an encrypted version of the data to the data memory **106**.

FIG. **3** shows a block diagram of an example of a computer system **300** that can include the CPU **102**, the instruction address/instruction bus **103**, the instruction memory **104**, the data address/data bus **105**, the data memory **106**, or a peripheral device **320**. The CPU **102** can include one or more Transformation eXecution Engines (TXE) **322**A-D. The TXEs **322** can be configured to receive instructions and data in a Harvard architecture format and execute the instructions as a function of the data. The data can be received from the data memory **106** or a peripheral device **320**. The TXE **322** can be configured to execute legacy instructions received as a function of corresponding legacy data received. The TXE **322** can be configured to execute eight-bit code such as 1802 or Z80 8-bit code, 16-bit code, such as x86 code, 32-bit code, 64-bit code, or other bit codes. The TXE **322** can be configured to execute instructions coded in a specific language, such as Java, C, C++, Python, or Matlab, among others. The TXE **322** can provide application separation, execution assurance, or security between applications, such as by acting as a separation kernel.

The peripheral device **320** can be electrically or communicatively coupled to the CPU **102**. The peripheral device **320** can be a sensor, video display or recording, audio transmission or recording, transceiver, Geographical Positioning System (GPS), or other peripheral device.

FIG. **4** shows a block diagram of another computer system **400** with a tamper or malware resistant architecture. The computer system **400** can include a CPU **102**, the instruction memory address and control logic **423**, the data memory address and control logic **425**, a CCU **107**, micro-code module **426**, a computer control unit address, data, and control interconnect **434**, an instruction address/instruction bus **103**, or a data address/data bus **105**.

The CPU **102** can include a Harvard computer engine **428**, I/O **110**, a combination encryptor and decryptor **218**, an interrupt handler **430**, or an endian translation module **432**. The Harvard computer engine **428** can be configured to receive data and execute instructions in a Harvard architecture format. In one or more embodiments the Harvard computer engine **428** can be a TXE **322**.